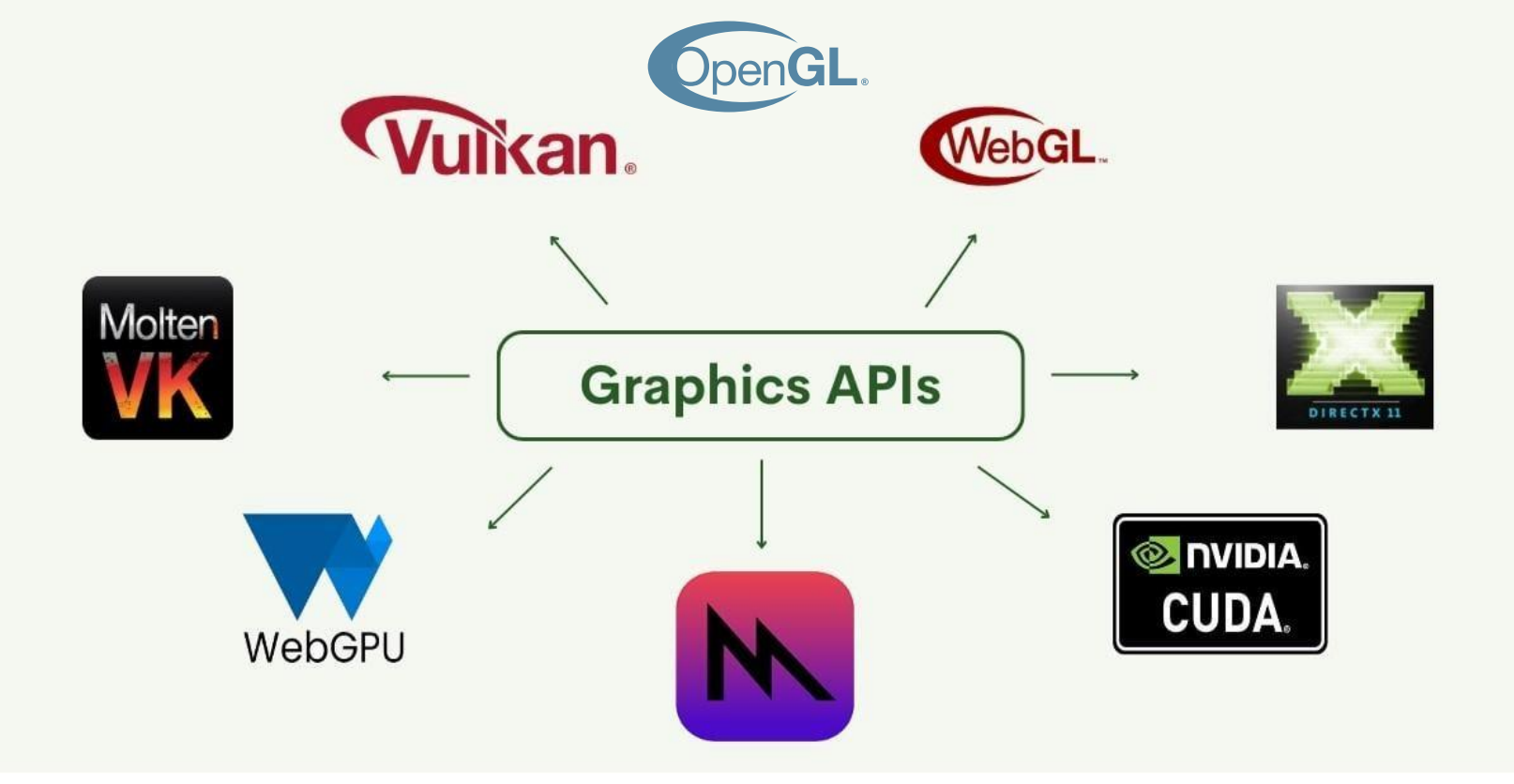
Insper

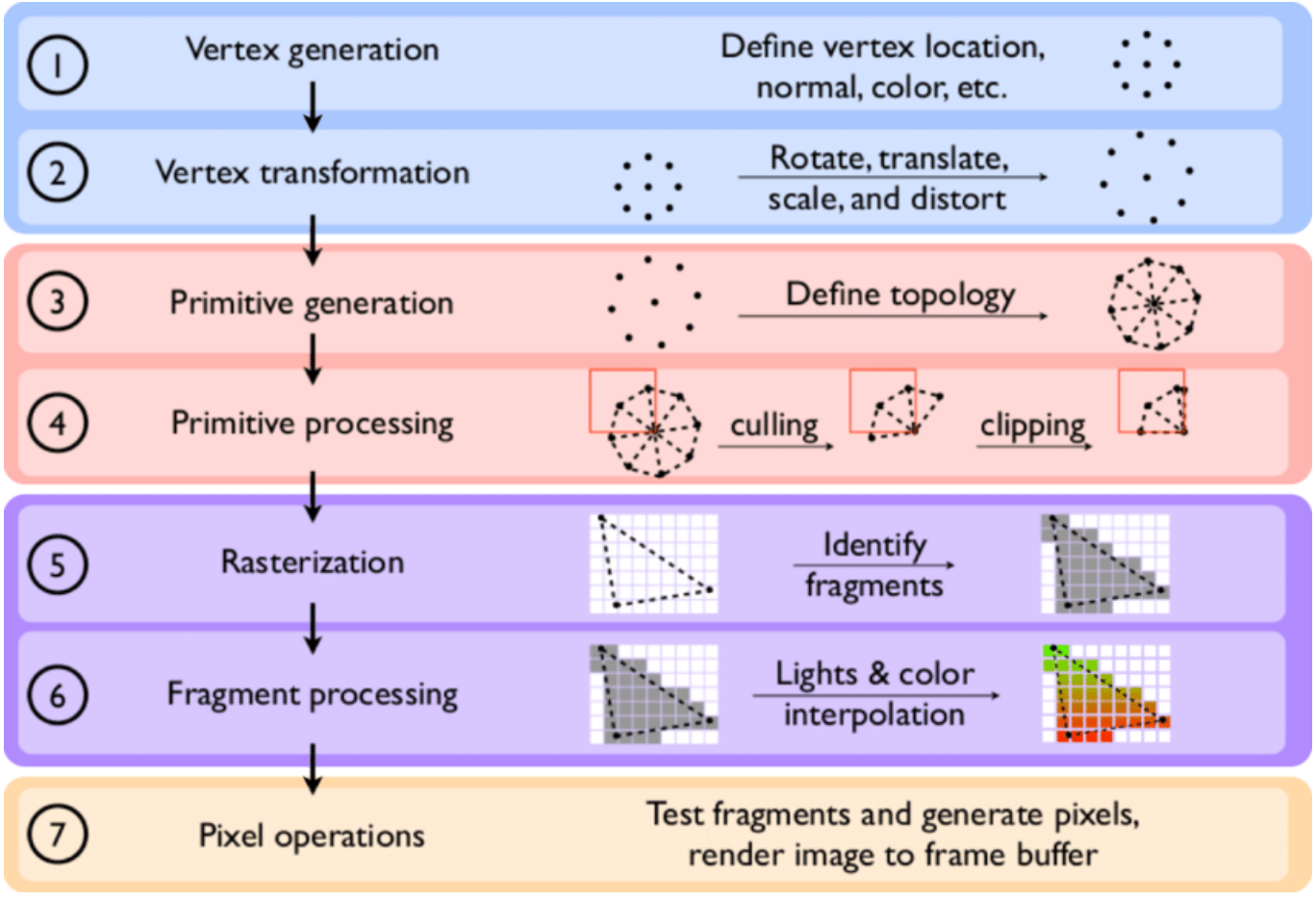
Computação Gráfica

Pipeline Gráfico & Shaders 1

Pipeline de Rasterização

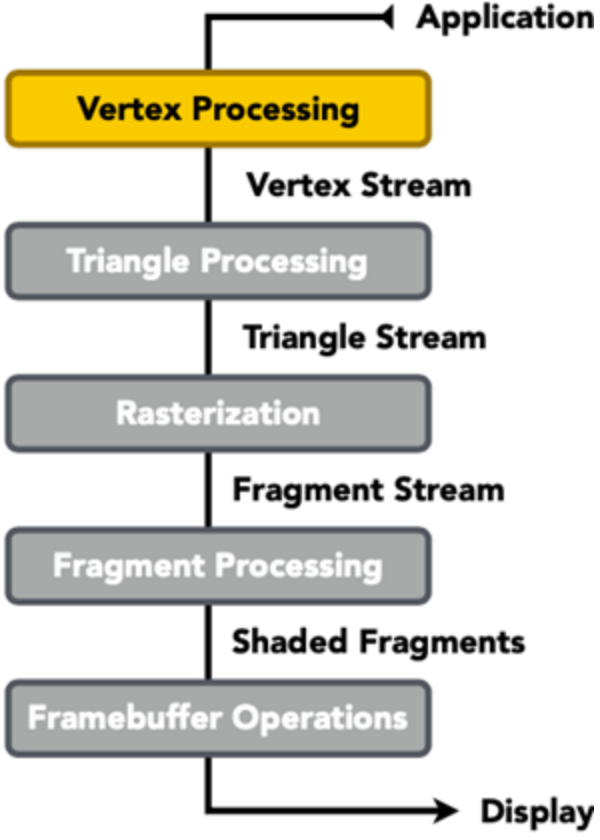


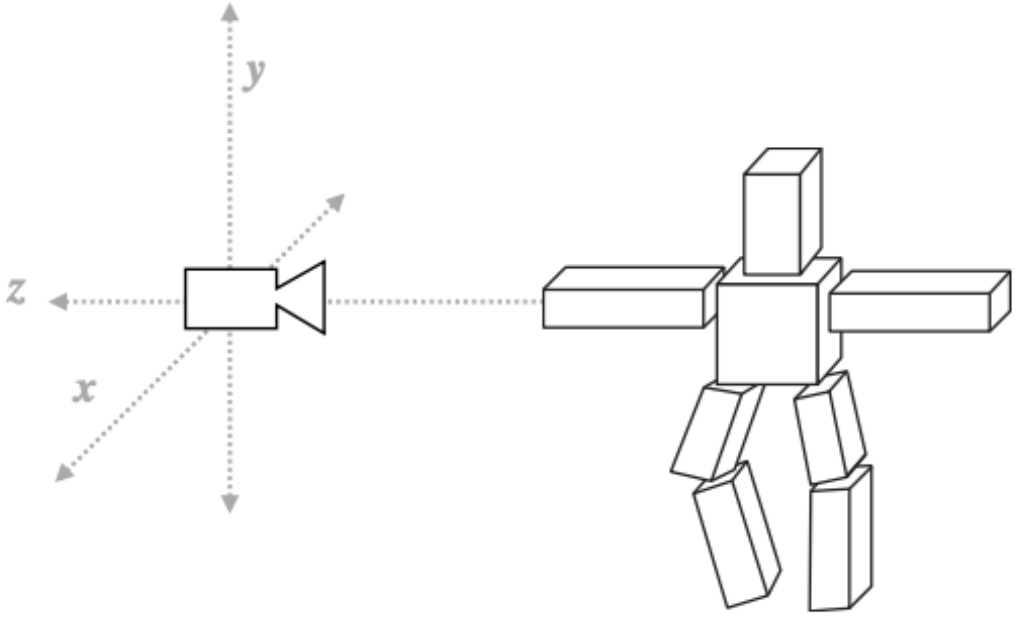
Pipeline de Rasterização



Insper

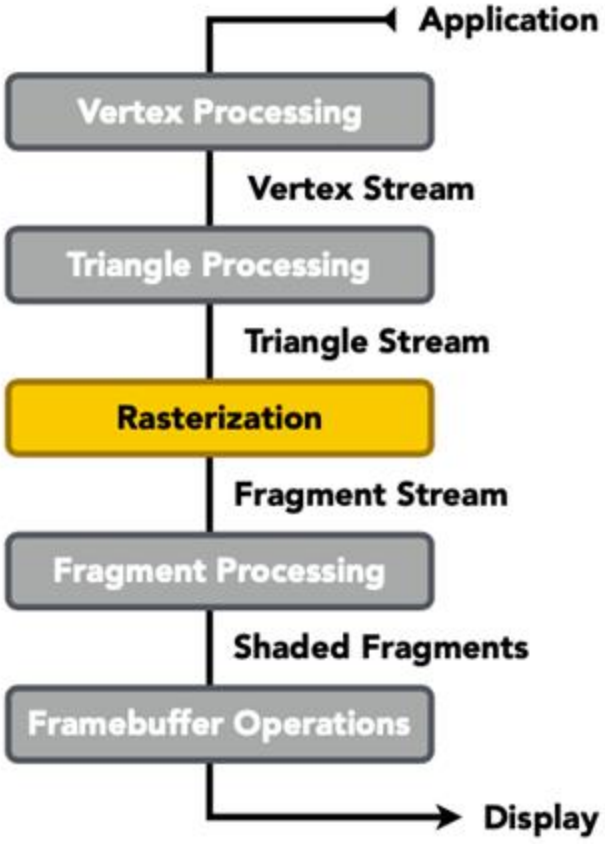
Transformações Geométricas e Visualização

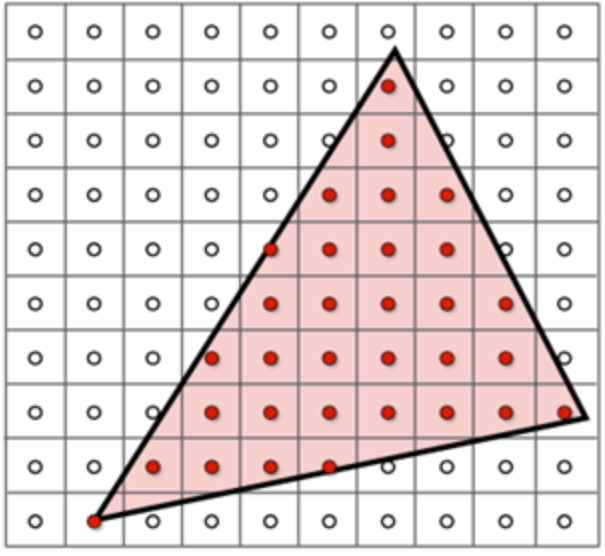




Insper

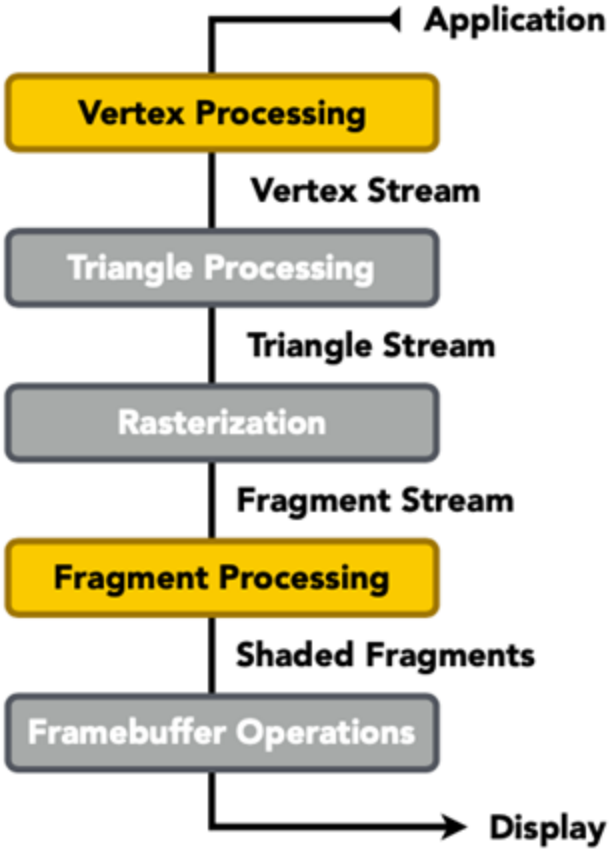
Amostrando os Triângulo

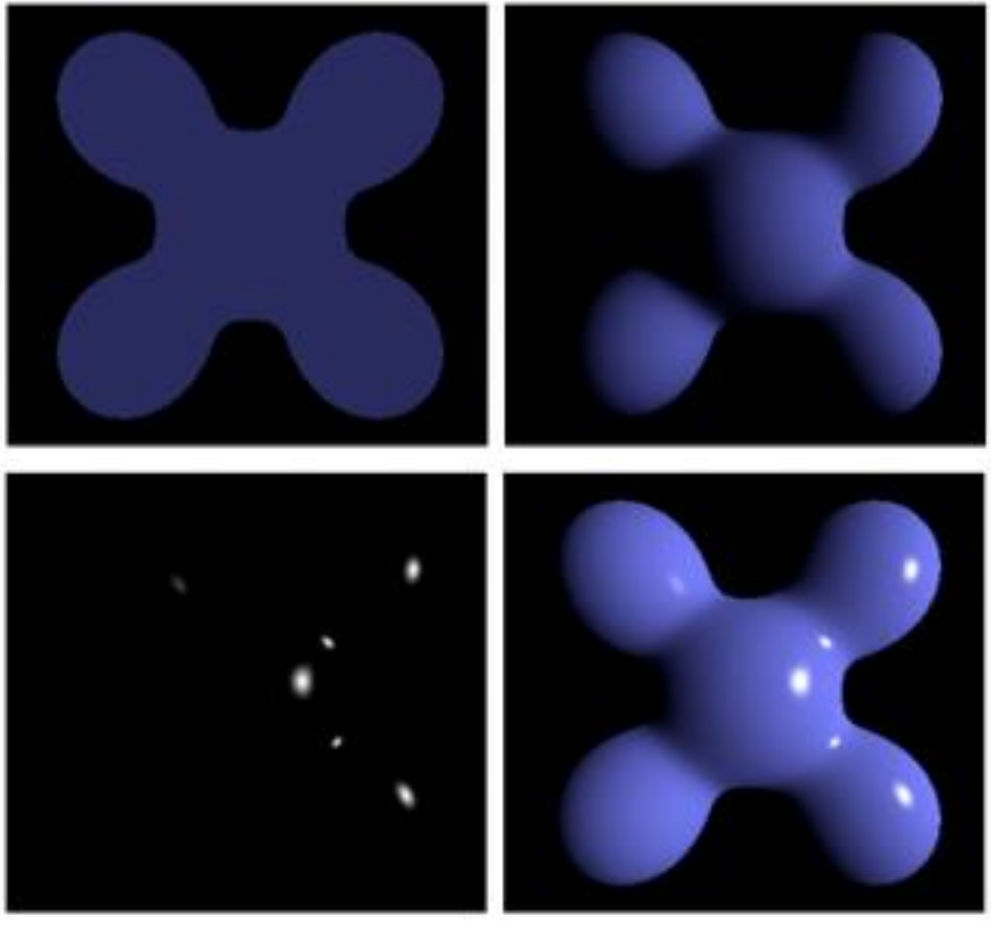




Insper

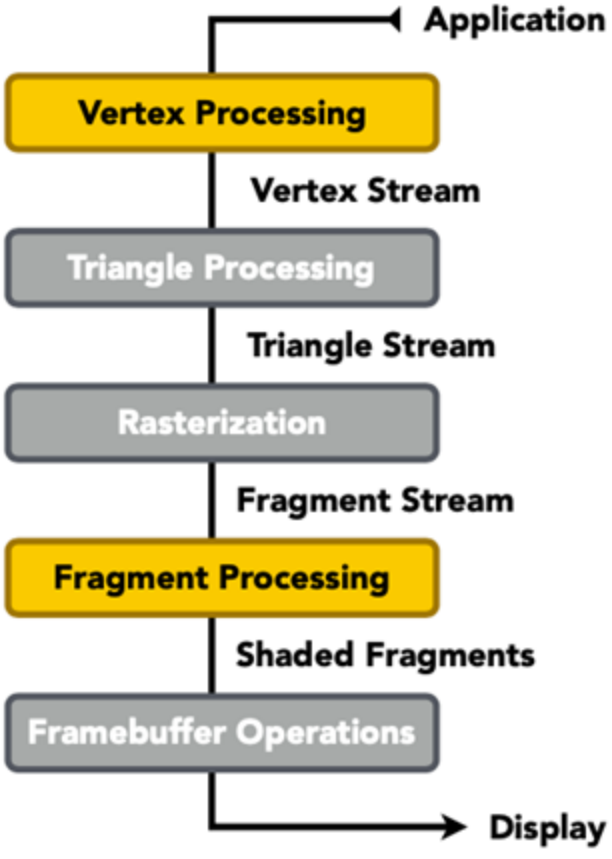
Avaliando a Função de Shading

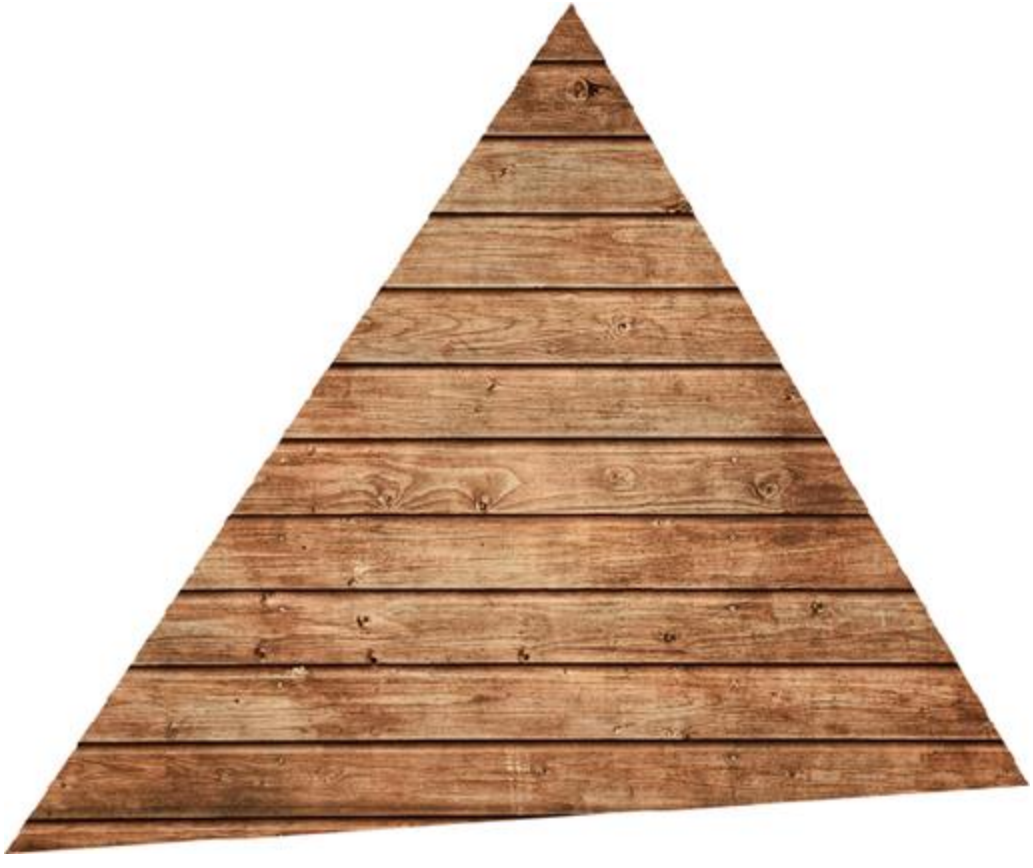




Insper

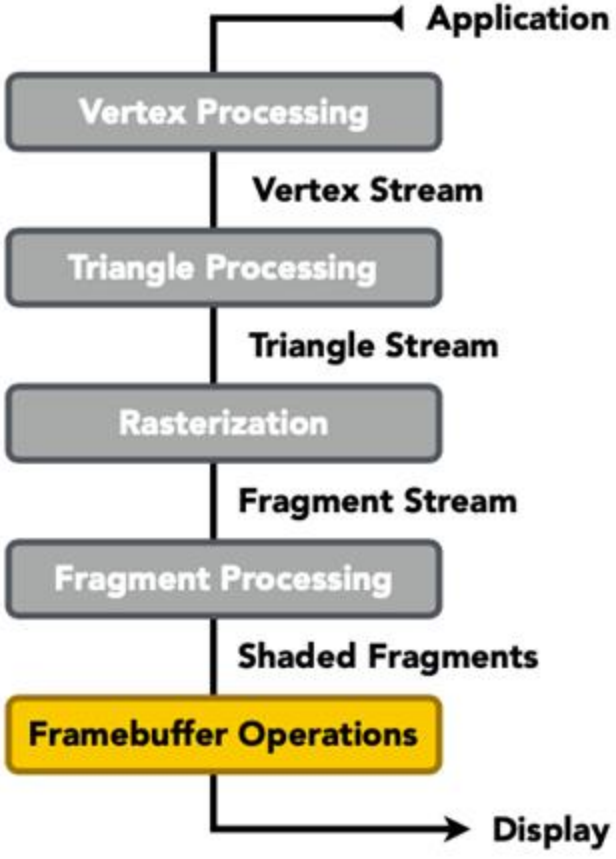
Mapeamento de Texturas





Insper

Teste de Visibilidade do Z-Buffer



Insper

Objetivo: Cenas 3D complexas em tempo real

Centenas de milhares a milhões de triângulos em uma cena
Cálculos complexos de vértices e fragmentos nos shaders
Alta resolução (2-4 megapixels + supersampling)
30-60 quadros por segundo (ainda mais alto para VR)

Unrecord - Official Early Gameplay Trailer

Insper

Pipeline de Rasterização: GPU

Para paralelizarmos as operações da pipeline gráfica, utilizamos a GPU ao invés da CPU, já que ela possui um número de processadores lógicos muito superior;

21.760 CUDA cores
NVIDIA GeForce RTX 5090

96 threads
AMD Ryzen Threadripper PRO 9995WX

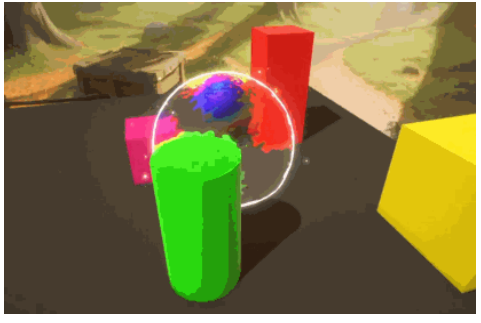
Pipeline de Rasterização: Papel da CPU

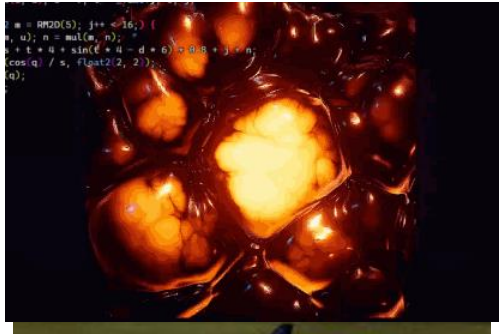
- A **CPU** é responsável por:
  - Enviar dados para a GPU, além de gerenciar o pipeline gráfico e a memória (buffers).
  - Definir o que a GPU irá renderizar, controlando movimentos de objetos e jogadores em cena.
    - Realizar, na maioria dos casos, a **detecção de colisões**.
    - Executar técnicas como **culling** de objetos e seleção de **nível de detalhe (LOD)**.
  - Outras funções mais intuitivas, como **comunicação multiplayer**, **áudio** e **efeitos sonoros**.

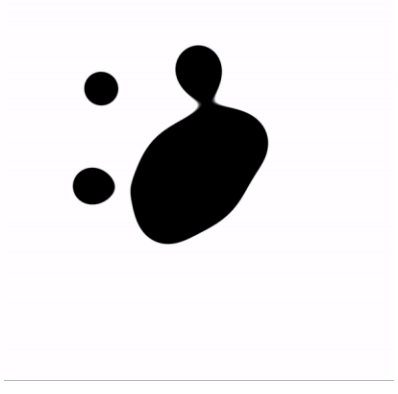
Pipeline de Rasterização: GPU

Shaders: O que são shaders?

Códigos desenvolvidos para execução em cada core da GPU, geralmente com foco em processamento visual.







Shaders

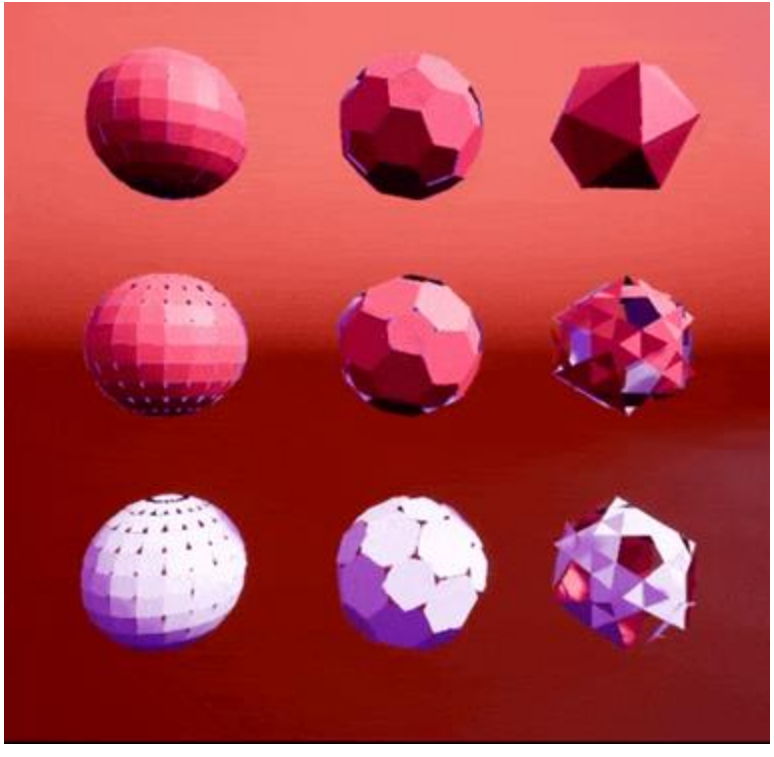
Linguagens para shaders mais populares são:

- **GLSL (OpenGL Shading Language)** → usada com **OpenGL** e **Vulkan** (via SPIR-V).
- **HLSL (High Level Shading Language)** → usada no **DirectX**.
- **MSL (Metal Shading Language)** → usada pela **API Metal** (Apple).
- **Cg (C for Graphics)** → linguagem da NVIDIA (obsoleta, mas influenciou bastante HLSL/GLSL).
- **SPIR-V** → não é uma linguagem de alto nível, mas um formato intermediário usado em **Vulkan** e compiladores.
- **WGSL (WebGPU Shading Language)** → usada pelo **WebGPU**, mais recente.

No curso, vamos aprender GLSL e WGSL.

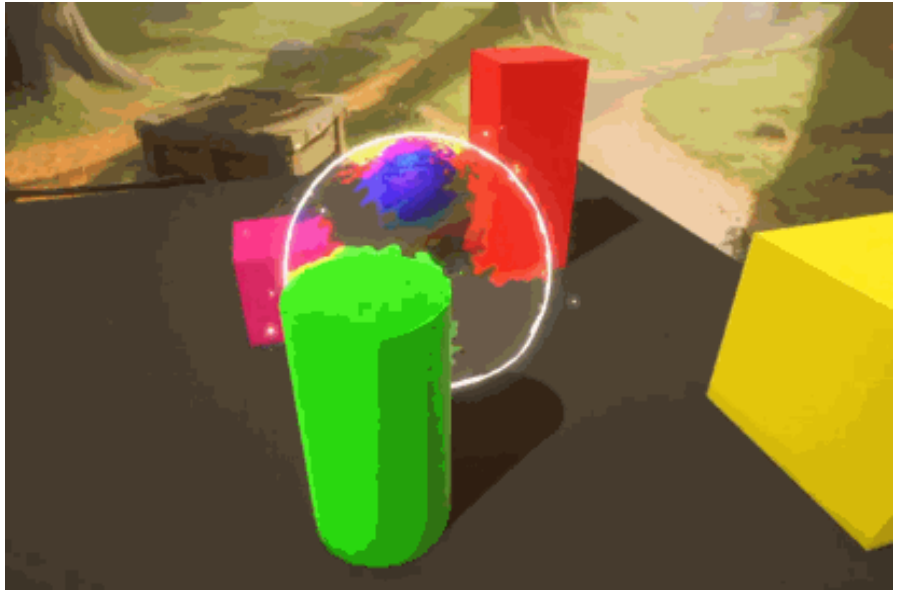
Shaders: Tipos de shaders

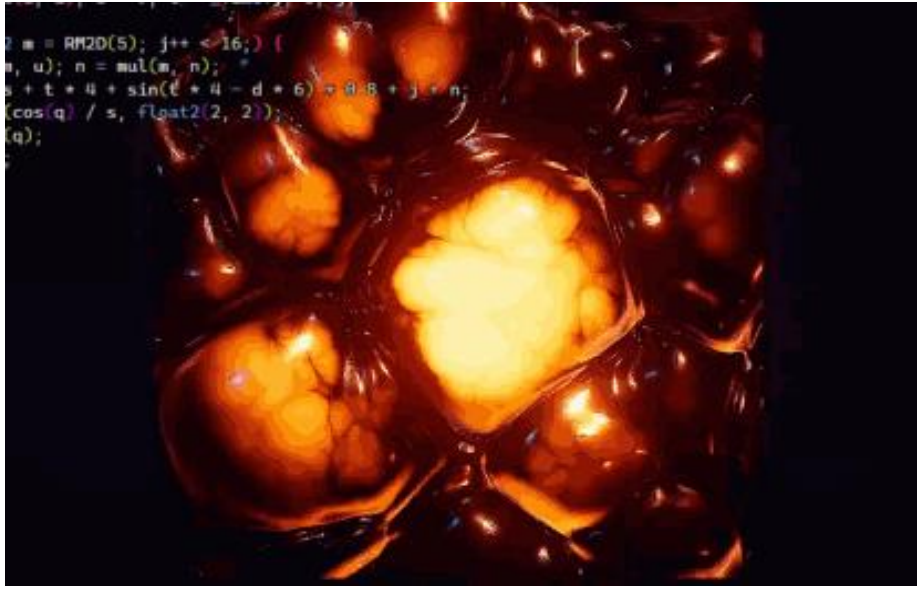
Vertex Shader: Manipula vértices dos objetos, gerando transformações e efeitos visuais.



Shaders: Tipos de shaders

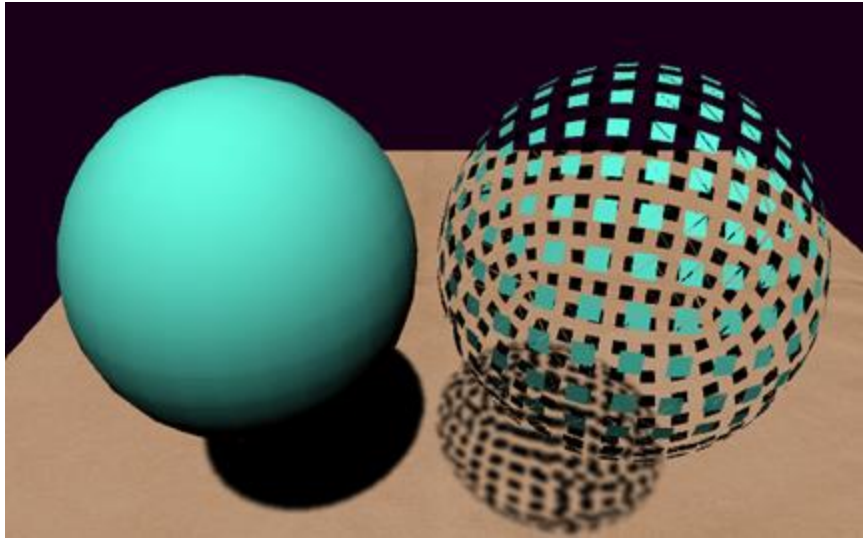
Fragment/Pixel Shader: Retorna a cor de cada pixel na tela com base nas informações recebidas do vertex shader ou de variáveis personalizadas.





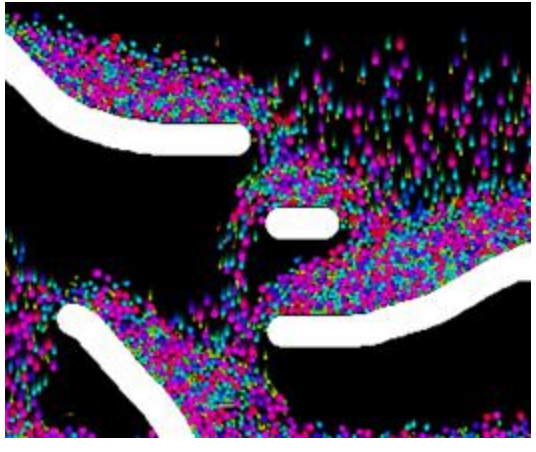
Shaders: Tipos de shaders

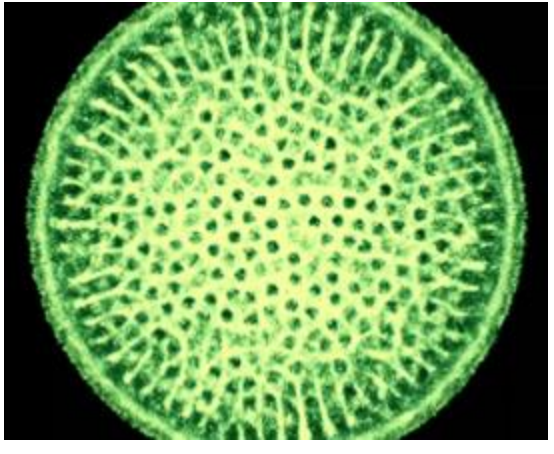
Geometry Shader Opera sobre primitivas geométricas (pontos, linhas, triângulos) adicionando ou removendo vértices, gerando novas primitivas e realizando outras operações que modificam a geometria.



Shaders: Tipos de shaders

Compute Shader: São programas executados na GPU voltados para cálculos gerais em paralelo, que não estão diretamente ligados ao pipeline gráfico tradicional, nem necessariamente ligados a visual. Este shader é muito utilizado hoje em dia em jogos, por ser extremamente customizável, genérico e baixo nível.



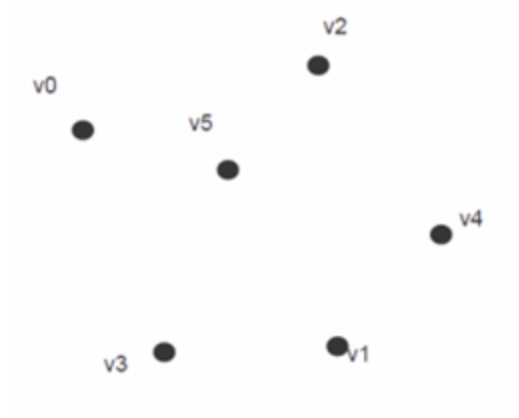


Shaders: Tipos de shaders

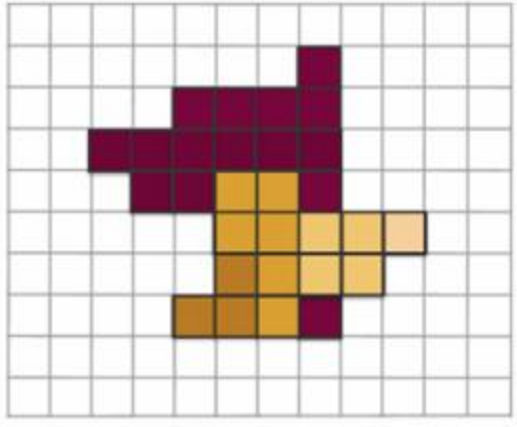
No curso vamos utilizar Fragment/Pixel shaders e compute shaders

Insper

Como os Shaders são invocados



Vertex Shader
invocado 6 vezes



Fragment Shader
invocado 35 vezes

(para os fragmentos
ocultos também)

fonte: https://eclass.aueb.gr/modules/document/file.php/INF143

Shaders Programáveis

Estágios de processamento de vértice e fragmento do programa
Descrever a operação para um único vértice (ou fragmento)

Exemplo de programa de shader em GLSL

```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader() {
   vec3 kd;
   kd = texture(myTexture, uv);
   kd *= clamp(dot(-lightDir, norm), 0.0, 1.0);
   gl_FragColor = vec4(kd, 1.0);
}
```

A função é executada uma vez por fragmento.

Exibe a cor da superfície na posição de amostra da tela do fragmento atual.

Este *shader* executa uma pesquisa de textura para obter a cor do material da superfície no ponto e, em seguida, executa um cálculo de iluminação difusa.

Compilação de um Shader

1 fragmento de entrada não processado

```
sampler mySampler;
Texture2D<float3> myTexture;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTexture.Sample(mySampler, uv);
  kd *= clamp ( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

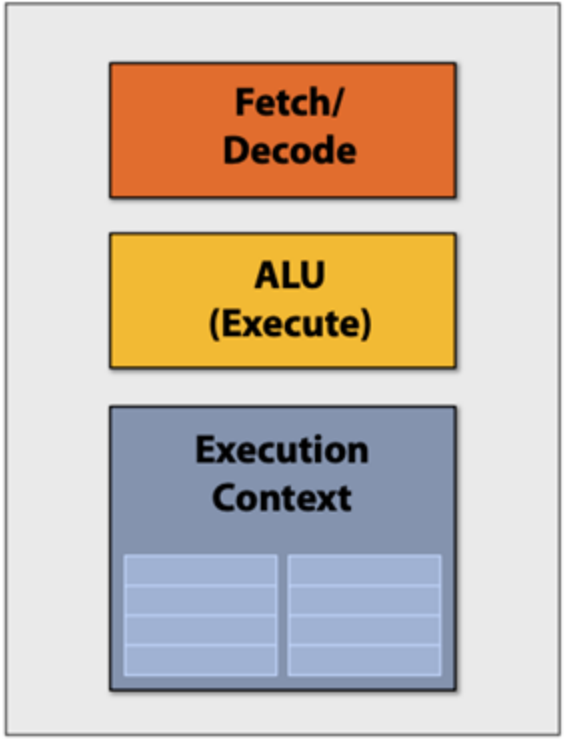
```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

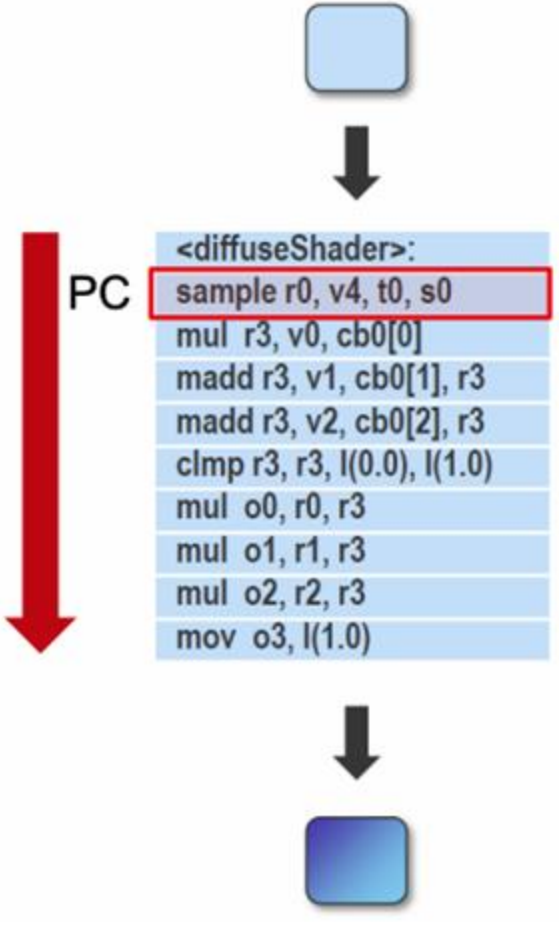
1 fragmento de saída processado

fonte: https://eclass.aueb.gr/modules/document/file.php/INF143

Mapeamento do Shader no HW

Execute o Shader em um único core:

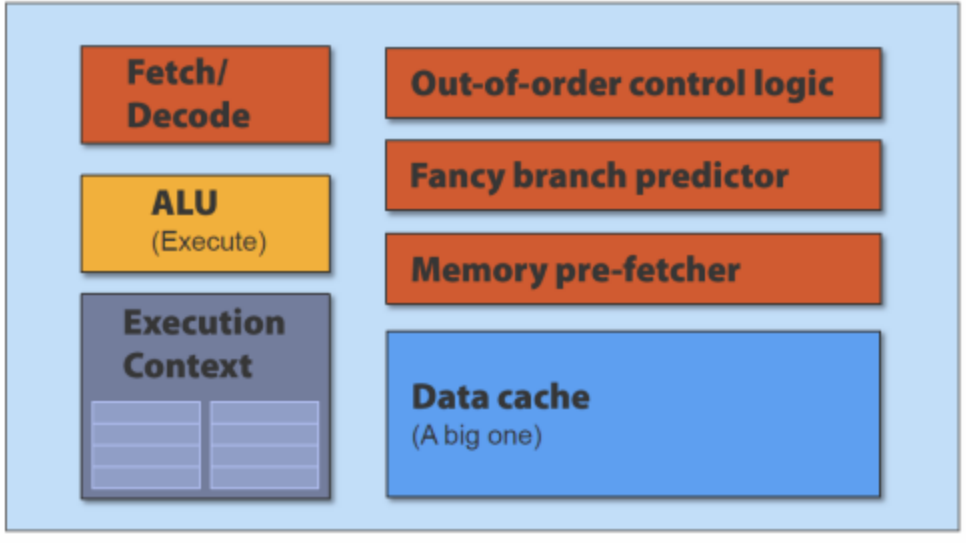




Insper

Mapeamento do Shader no Hardware

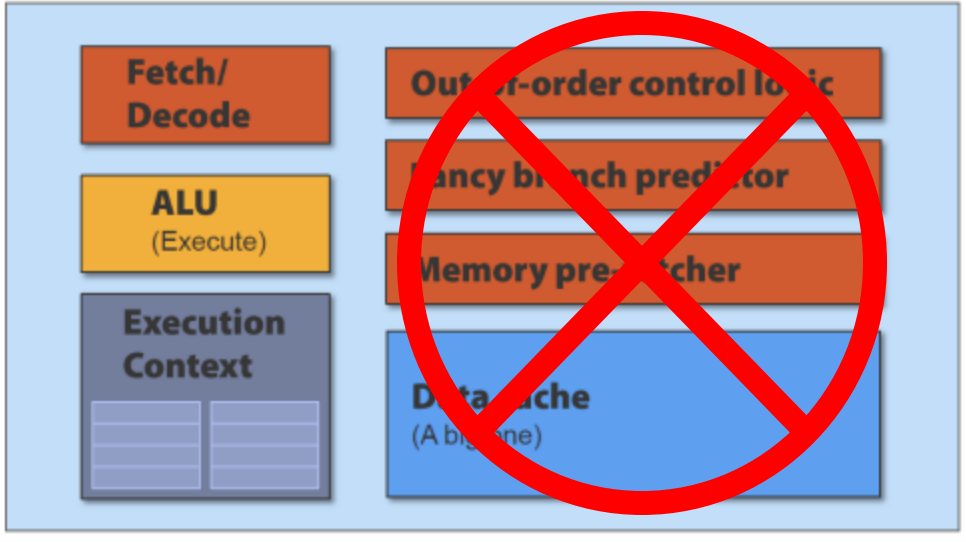
Um *core* de CPU



- Otimizado para acesso de baixa latência aos dados em cache
- Lógica de controle para execução fora de ordem e especulativa
- Grande cache L2

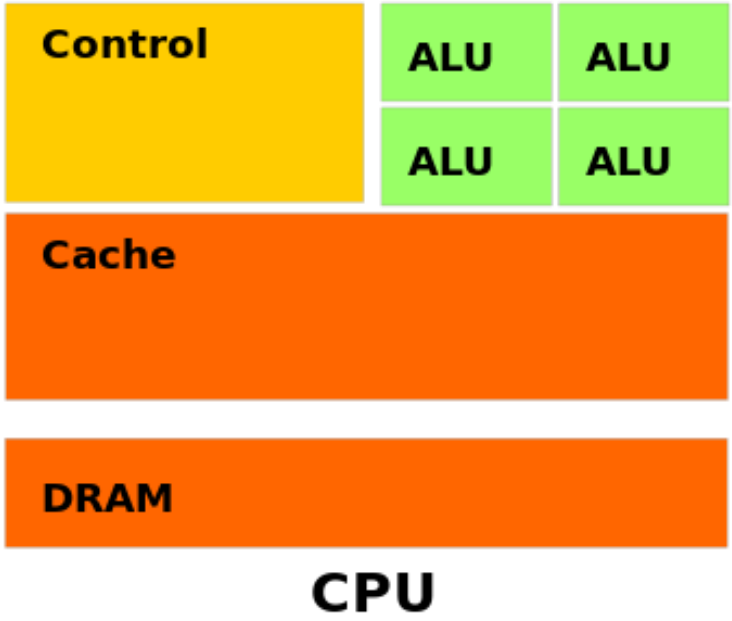
Reduzindo os Cores

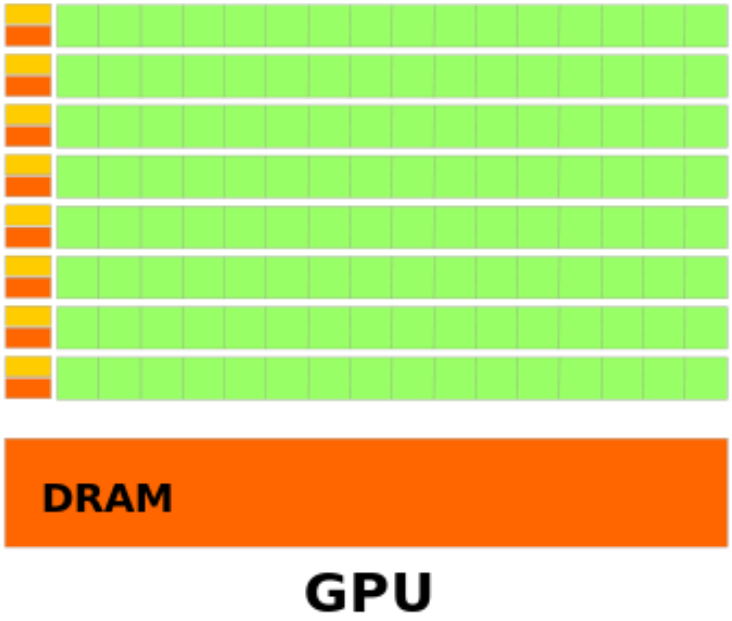
Um *core* de GPU



- Otimizado para computação paralela de dados
- Arquitetura tolerante à latência de memória
- Mais cálculos por mais transistores em ALUs
- Redução dos circuitos principais de controle

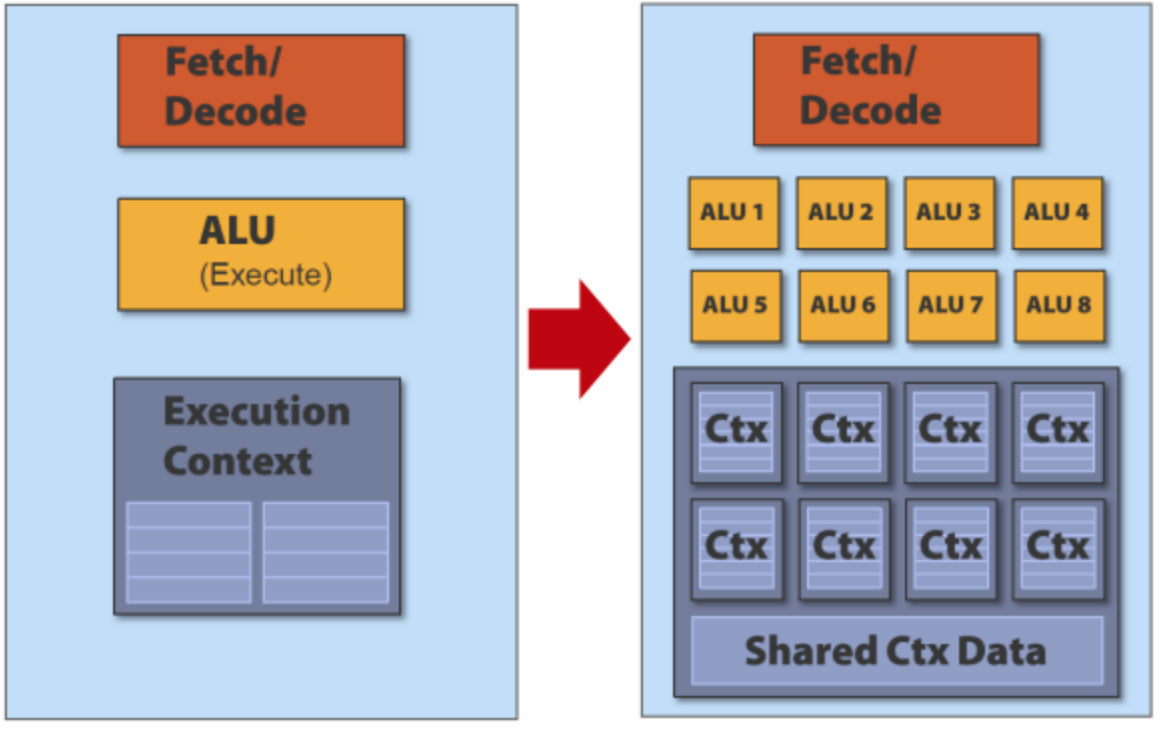
GPU architecture





Múltiplos dados (SIMD)

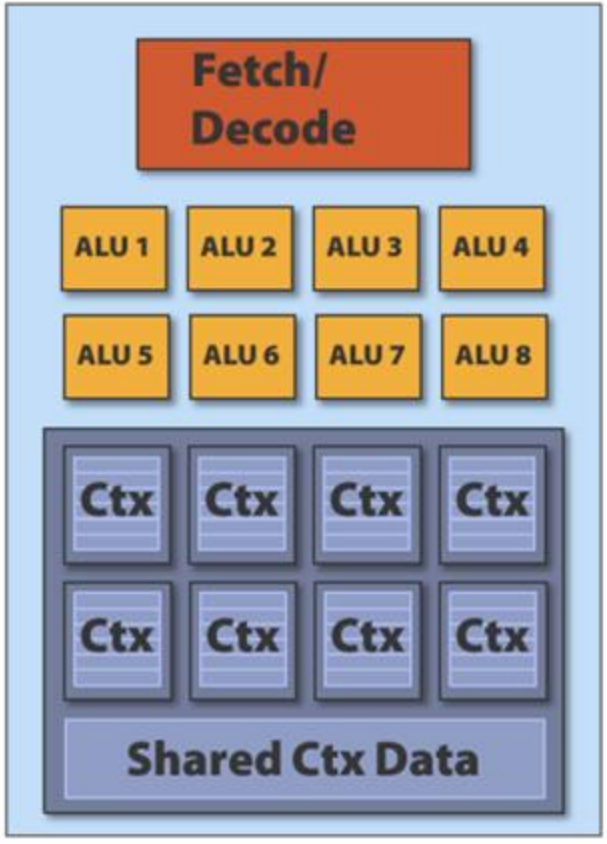
Shaders são inerentemente executados muitas vezes, repetidas vezes em vários registros de seus fluxos de dados de entrada.

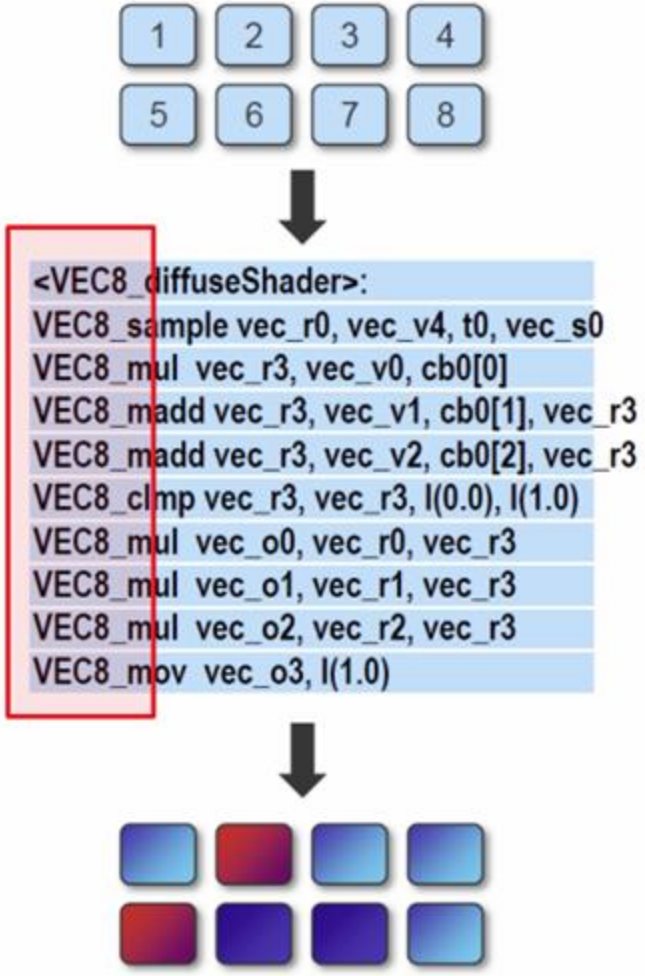


Amortize o custo / complexidade do gerenciamento de instruções para várias ALUs.

Compartilhe a unidade de instrução.

SIMD Cores: Vectorized Instruction Set

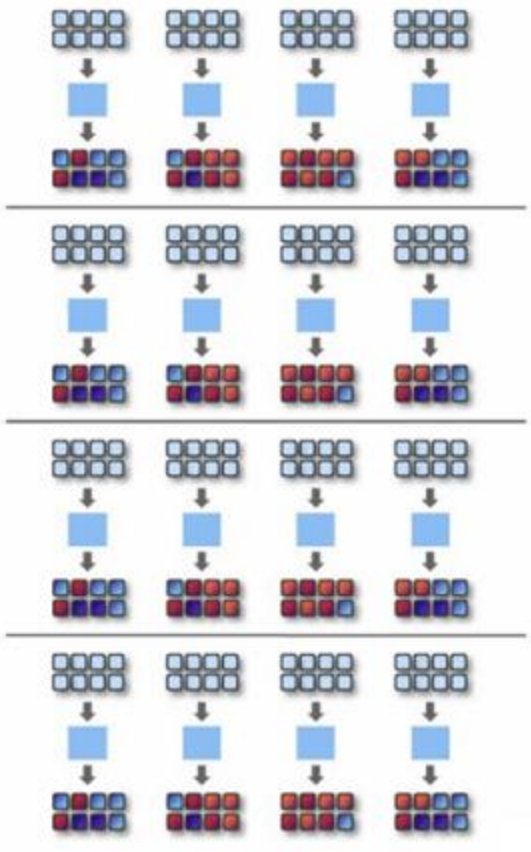


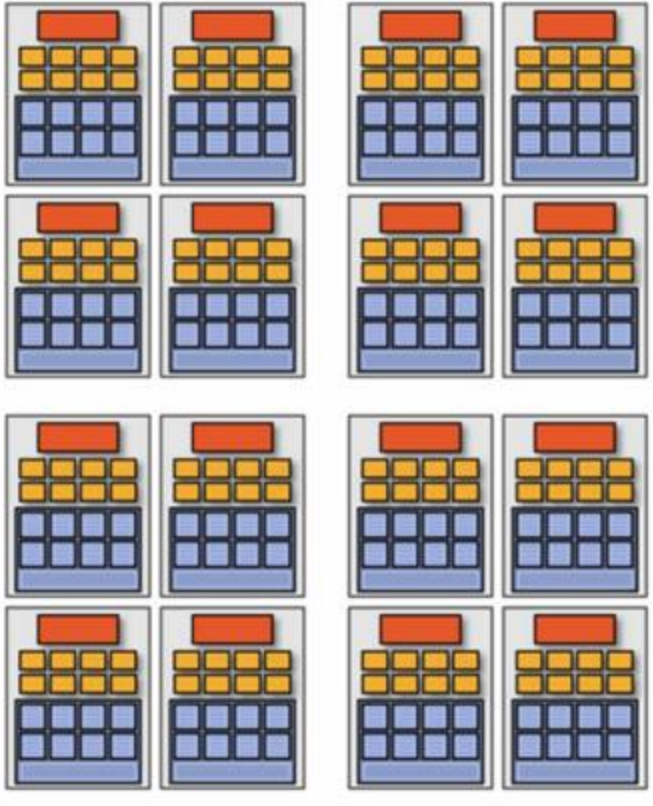


Insper

Adicionando tudo: vários núcleos SIMD

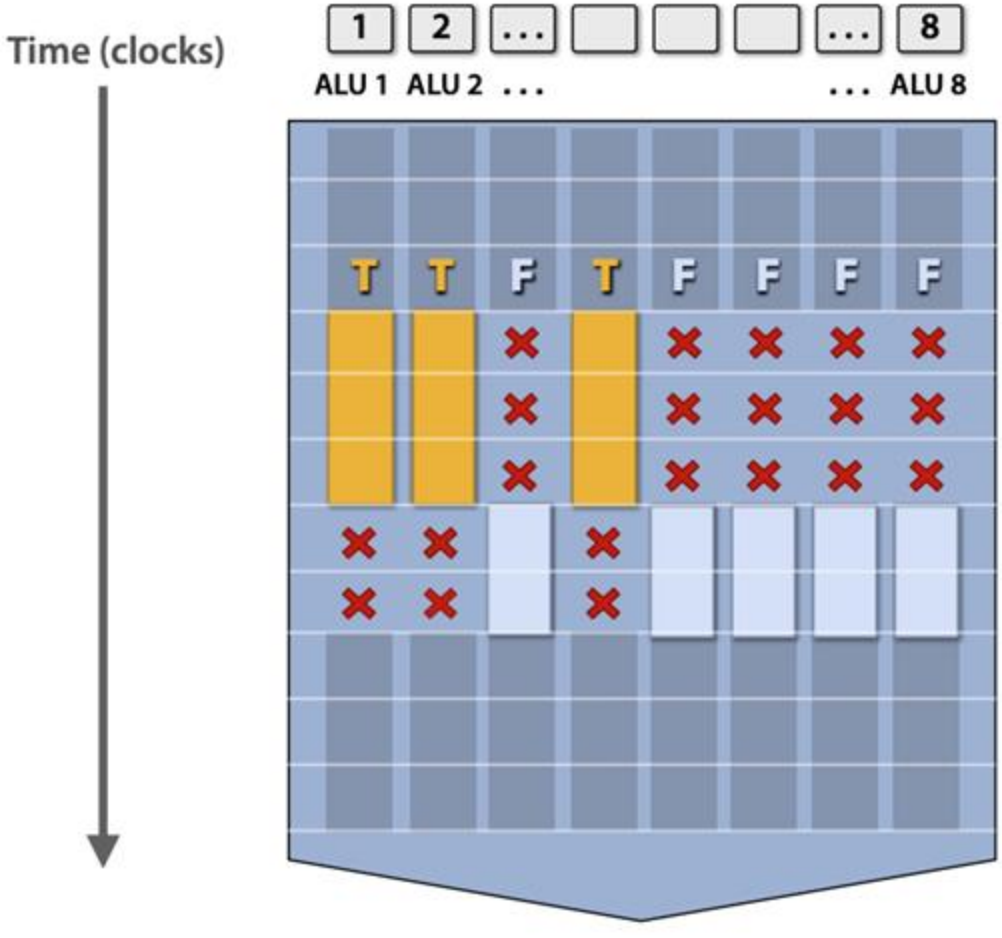
Neste exemplo: 128 dados processados simultaneamente





16 cores = 128 ALUs = 16 fluxos de instruções simultâneos

Condicionais / Branches



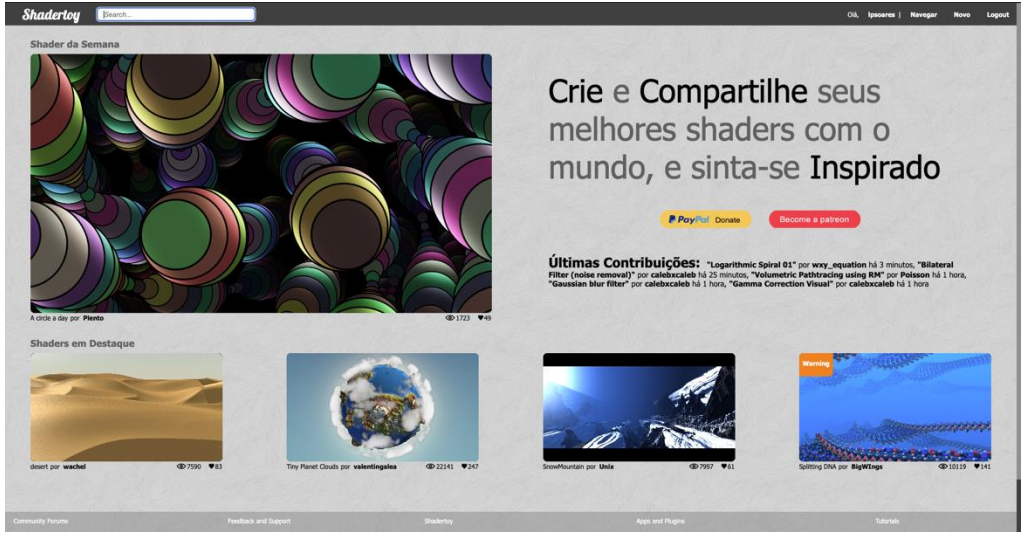

```
<unconditional
 shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}
<resume unconditional
 shader code>
```

Shadertoy

O Shadertoy é uma ferramenta da internet que permite escrever Fragment Shaders direto no navegador.

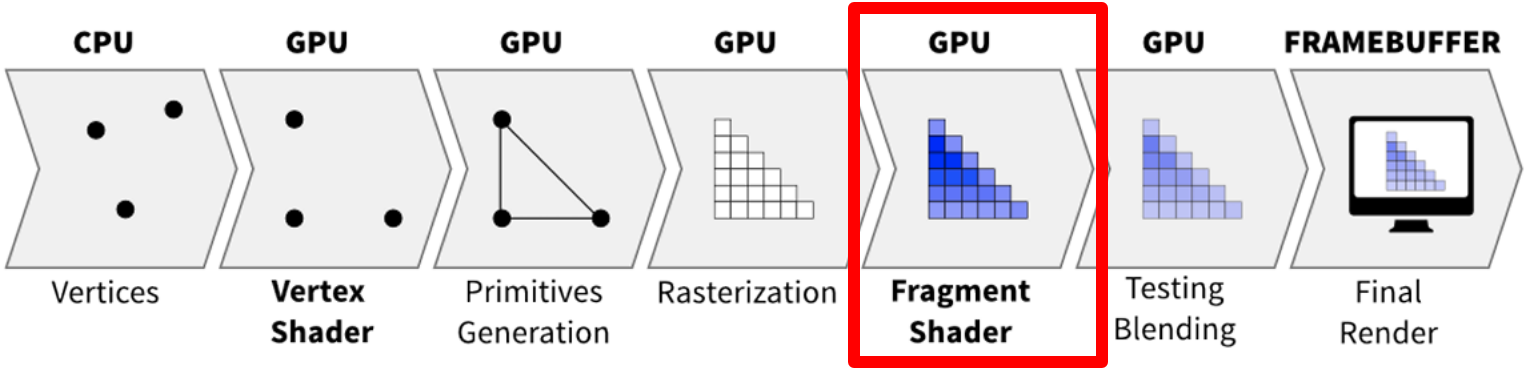
Alguns Uniforms já são automaticamente fornecidos, e todo o processo de compilação é basicamente instantâneo.

O Shadertoy usa alguns padrões para passar os dados, como no caso a chamada do main(), que é mainImage().



Insper

Fragment Shader



O Shadertoy não permite que você escreva vertex shaders e apenas permite que você escreva fragment shaders. Essencialmente, ele fornece um ambiente para experimentar e desenvolver no fragmento shader, tirando todo o proveito do paralelismo de pixels na tela.

Imagem:F.Andreussi(BUW)

Live Coding

Imagem:F.Andreussi(BUW)

Projeto

Ainda precisamos das duas próximas aulas

Recomendação para testar com sua musica favorita: Chrome extension

Imagem:F.Andreussi(BUW)

Projeto: Rubrica

C

- Circunferência reagindo a música **(FFT ou amplitude global)**;
- Retângulos em volta, reagindo a musica (devem ser posicionados seguindo a curva da circunferência **(Obrigatório reagir ao FFT)**. Cada retângulo deve reagir à um canal do áudio;

C+

- Qualquer tipo de aleatoriedade:
  - Grossura dos retângulos;
  - Cores;
  - Alterar resposta da frequência da musica;
  - Efeito de background;
- Glow/Neon (necessário utilizar uma SDF 2D de retângulo);
- Controle de cores;

Adiciona meio conceito

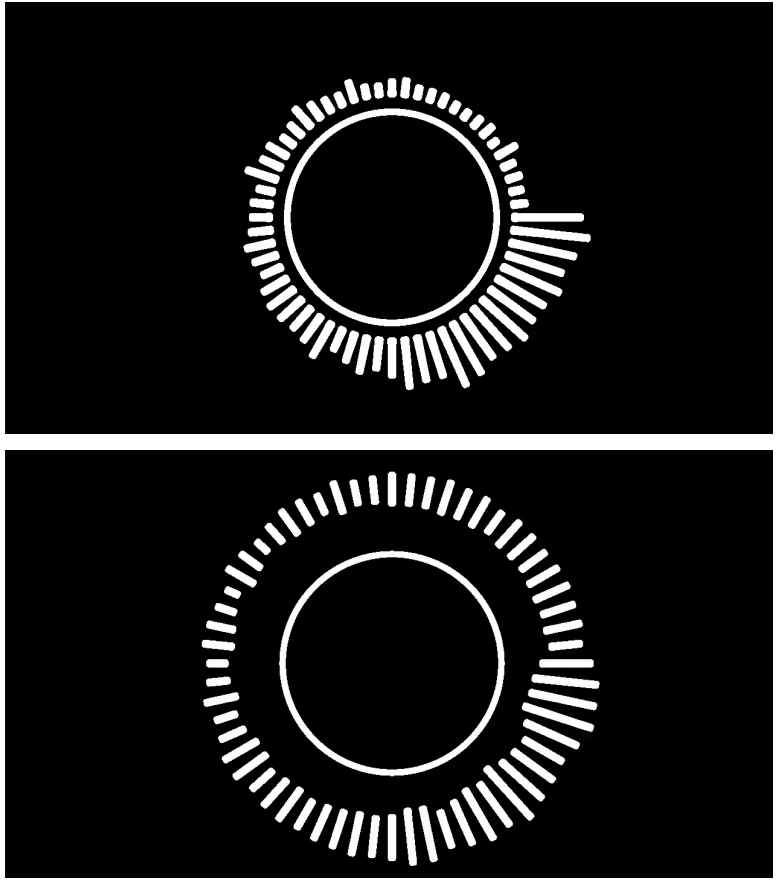
- Adicionar outro efeito como starfield / Warp drive effect;
- Efeito de câmera shake ou qualquer tipo de UV displacement;
- Adicionar outra forma geométrica que reage a música;
- Tratar a entrada de áudio: Suavizar a entrada, aplicar algum filtro de audio;
- Criação de paletas diferentes proceduralmente: https://iquilezles.org/articles/palettes/;

FFT (Fast Fourier Transform)

Imagem:F.Andreussi(BUW)

Projeto: Rubrica

Exemplos C



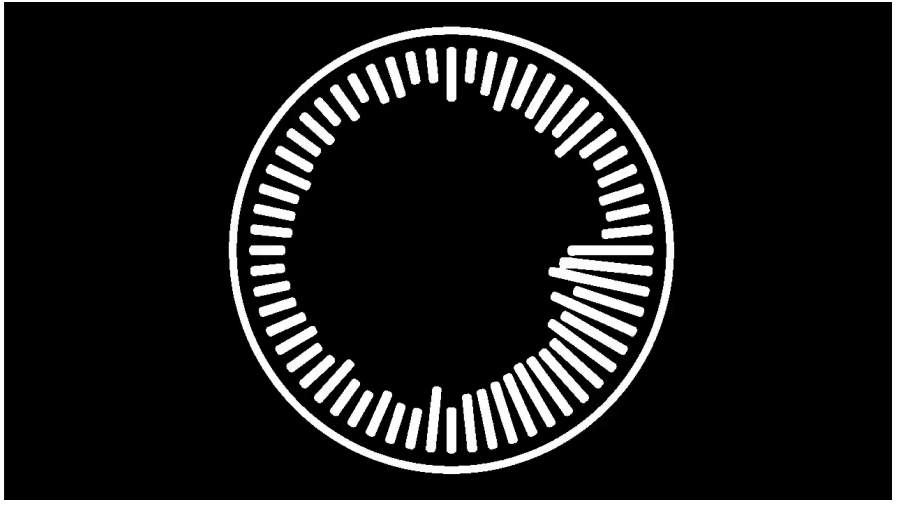


Imagem:F.Andreussi(BUW)

Projeto: Rubrica

Exemplo C+

Imagem:F.Andreussi(BUW)

Projeto: Rubrica

Exemplo A+

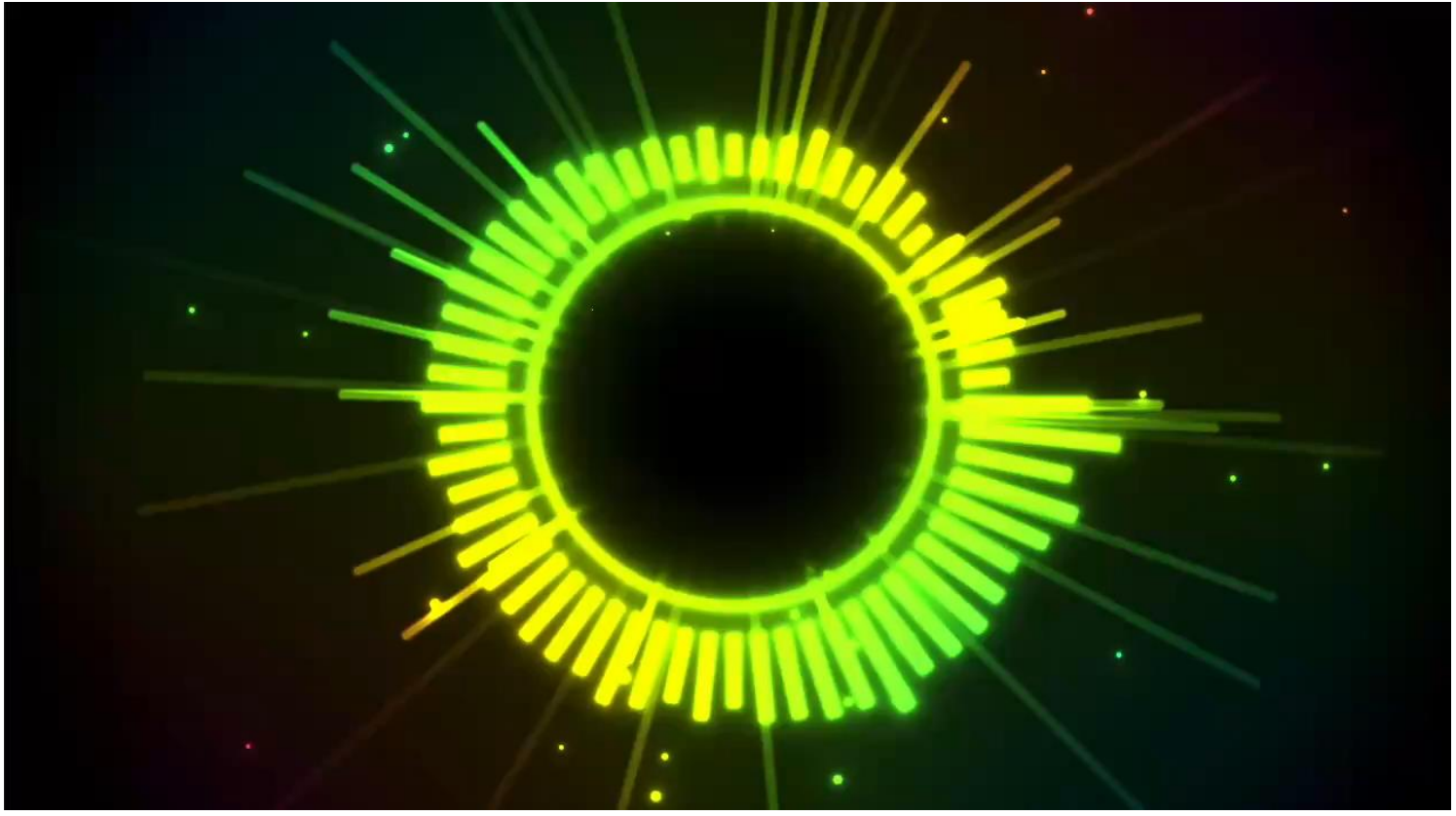


Imagem:F.Andreussi(BUW)

Projeto: Dicas

Passo a passo sugerido

C

- Desenhar a circunferência;
- Distribuir retângulos ao longo da circunferência;
- Capturar espectro (FFT) ou amplitude global;
- Utilizar os canais capturados para alterar as formas geométricas

C+

- Dividir o output por um numero baixo (glow)
- Utilizar um valor aleatório (0-1) para controlar alguma etapa do código
- Multiplicar a máscara gerada com as figuras por uma paleta de cores ou uma cor

Imagem:F.Andreussi(BUW)

Projeto: Dicas

Links do código da aula e outros:

Documentação GLSL: https://docs.gl/sl4/all
Aula: https://www.shadertoy.com/view/WcfcDf
Áudio: https://www.shadertoy.com/view/Xds3Rr

Insper

Computação Gráfica

Luciano Soares
<lpsoares@insper.edu.br>

Fabio Orfali
<fabioo1@insper.edu.br>

Gustavo Braga
<gustavobb1@insper.edu.br>